Introduction to Object-Oriented Programming

Week 9

MEMS 1140—Introduction to Programming in Mechanical Engineering

Learning Objectives (L.O.)

At the end of this lecture, you should understand/be able to:

- ❏ What the philosophy of Object-Oriented Programming is;
- ❏ What classes are;
- ❏ Define a class;
- ❏ Define objects from a class;
- ❏ What objects are and how they differ from classes;

Table of Contents (ToC)

1 – Functional vs. Object-Oriented

So far, we have practiced "**functional**" programming: each piece of code is written to accomplish a specific task.

This lecture introduces **object-oriented** programming: we focus on representing the data in our systems and *then* define the functions that can operate on that data.

The notable difference: OOP restricts functions to operating *only* on the data to which they are tied.

2 – What Classes Are

This is accomplished by defining **classes**.

A class is the *blueprint* for how an **object** of that type is structured.

The class definition organizes **`properties`** and **`methods`** to enforce rules about which functions can access which data.

Let's illustrate this next with an example of material elongation due to axial loading.

3 – Defining the Class Structure


```
classdef MaterialSample
    .
    .
    .
    .
        .
            .
        .
        .
            .
        .
    .
end
```

3 – Defining Class Properties


```
classdef MaterialSample
  properties
    E  .  .
  end
   .
     .
       .
     .
     .
       .
     .
   .
end
```

3 – Defining Class Properties


```
classdef MaterialSample
  properties
    E, A  .
  end
  .
    .
      .
    .
    .
      .
    .
  .
end
```

3 – Defining Class Properties


```
classdef MaterialSample
  properties
    E, A, L
  end
   .
     .
       .
     .
     .
       .
     .
   .
end
```

3 – Defining the Class Constructor


```
classdef MaterialSample
  properties
    E, A, L
  end
  methods
    function obj = MaterialSample(E,A,L)
      .          .          .
    end
    .
      .
    .
  end
end
```

3 – Defining the Class Constructor


```
classdef MaterialSample
  properties
    E, A, L
  end
  methods
    function obj = MaterialSample(E,A,L)
      obj.E = E; .          .
    end
    .
      .
    .
  end
end
```

3 – Defining the Class Constructor


```
classdef MaterialSample
  properties
    E, A, L
  end
  methods
    function obj = MaterialSample(E,A,L)
      obj.E = E; obj.A = A; .
    end
    .
      .
    .
  end
end
```

3 – Defining the Class Constructor


```
classdef MaterialSample
  properties
    E, A, L
  end
  methods
    function obj = MaterialSample(E,A,L)
      obj.E = E; obj.A = A; obj.L = L;
    end
    .
      .
    .
  end
end
```

3 – Defining the Axial Loading Method


```
classdef MaterialSample
  properties
    E, A, L
  end
  methods
    function obj = MaterialSample(E,A,L)
      obj.E = E; obj.A = A; obj.L = L;
    end
    function obj = axialLoading(obj, P)
      .
    end
  end
end
```

3 – Defining the Axial Loading Method


```
classdef MaterialSample
  properties
    E, A, L
  end
  methods
    function obj = MaterialSample(E,A,L)
      obj.E = E; obj.A = A; obj.L = L;
    end
    function obj = axialLoading(obj, P)
      obj.L = .
    end
  end
end
```

3 – Defining the Axial Loading Method


```
classdef MaterialSample
  properties
    E, A, L
  end
  methods
    function obj = MaterialSample(E,A,L)
      obj.E = E; obj.A = A; obj.L = L;
    end
    function obj = axialLoading(obj, P)
      obj.L = obj.L + .
    end
  end
end
```

3 – Defining the Axial Loading Method


```
classdef MaterialSample
  properties
    E, A, L
  end
  methods
    function obj = MaterialSample(E,A,L)
      obj.E = E; obj.A = A; obj.L = L;
    end
    function obj = axialLoading(obj, P)
      obj.L = obj.L + (P * obj.L) / (obj.E * obj.A);
    end
  end
end
```

4 – Using the Class

Instantiate an **object** of 316L Stainless Steel:

```
>> stainless_steel_316L = MaterialSample(193e9, 0.02^2, 1);
.
.
.
.
```

Command Window

This line uses the constructor to set values for **E**, **A**, and **L**.

4 – Using the Class

Apply the **`axialLoading`** function:

```
>> stainless_steel_316L = MaterialSample(193e9, 0.02^2, 1);
>> stainless_steel_316L = stainless_steel_316L.axialLoading(1e6);
.
.
.
```

Command Window

This line applies a force to the sample and, in turn, elongates it.

4 – Using the Class

Print the object's new length `L`:

```
>> stainless_steel_316L = MaterialSample(193e9, 0.02^2, 1);
>> stainless_steel_316L = stainless_steel_316L.axialLoading(1e6);
>> stainless_steel_316L.L
ans =
     1.0130
```

Command Window

This line accesses the instance's property `L` to print its value.

5 – Classes vs. Objects

Many people conflate classes and objects.

A **class** is a set of *instructions*. It's the recipe in a cookbook.

An **object** is the *manifestation* of those instructions. It's the actual meatloaf dish sitting on your kitchen counter.

You can't (satisfyingly) `eat` the cookbook recipe, but you *can* `eat` the meatloaf dish.

6 – Summary

This lecture covered:

- ✓ The philosophy of Object-Oriented Programming

  OOP is an approach to programming that wraps data together with the functions that operate on it in order to create an intentional usage structure for a codebase.

- ✓ What classes are

  A class is the blueprint for how attributes (data) and methods (functions) are wrapped together.

6 – Summary

- ✓ L.O.1
- ✓ L.O.2
- ✓ L.O.3
- ✓ L.O.4
- ✓ L.O.5

✓ Defining a class

The class definition is denoted by **`classdef <ClassName>`**. Then the names of its **`properties`** are listed out, followed by the **`methods`**, which can only operate on those properties. The first method should be a constructor, which specifies the procedure for how an object of that class is brought into existence.

6 – Summary

✓ Defining objects from a class

By declaring a variable equal to the class name according to the constructor, you create an instance of that class. Then you can apply any of the associated methods using dot notation.

✓ What objects are and how they differ from classes

While classes are the blueprint, an object is the actual instance in code that exists according to the way the class is structured.